Experiment 8

Student Name: Diwakar Kumar **UID: 22BCS10849**
Branch: CSE **Section/Group: 640-B**
Semester: 6th **Date of Performance: 31/03/25**
Subject Name: Cloud IoT Edge ML Lab **Subject Code: 22CSP-367**

1. **Aim:** Design a CNN based approach for vehicle recognition & traffic estimation based on IoT.

2. **Objective:** To create a cloud-based back-end for IoT applications by setting up Amazon EC2 servers with different operating systems.

3. **Hardware / Software Used:** IoT Cameras (CCTV, IP cameras, or edge devices), IOT Sensors (if integrated), Internet Connectivity, AWS Account, SSH-Enabled Device.

4. **Procedure:**
   1. Hardware Installation
      - Mount IoT cameras at the desired traffic monitoring points.
      - Connect additional sensors (if used) to the network or edge devices.
      - Ensure devices are powered and connected to the internet.
   2. Software Installation
      - Install Python and necessary libraries
      - Clone the project repository:
      - Configure the application settings in the config.py file.
   3. Deploy the System
      - For cloud-based deployment, configure the system on a cloud server. • For edge deployment, set up a local server and connect all devices to it.

5. **Code:**

```
# Mount Google Drive to access datasets
from google.colab import drive
drive.mount('/content/drive')
# Install necessary libraries
!pip install tensorflow opencv-python matplotlib
# Import libraries import
tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator import
matplotlib.pyplot as plt
import os
```

```
# Set dataset paths (update with your dataset location)
train_dir = '/content/drive/MyDrive/vehicle_dataset/train'
val_dir = '/content/drive/MyDrive/vehicle_dataset/validation'
# Data preprocessing and augmentation train_datagen
= ImageDataGenerator(
 rescale=1./255,
rotation_range=30,
 width_shift_range=0.2,
 height_shift_range=0.2,
 shear_range=0.2,
 zoom_range=0.2,
 horizontal_flip=True,
 fill_mode='nearest'
)
val_datagen = ImageDataGenerator(rescale=1./255)
# Load images in batches train_generator =
train_datagen.flow_from_directory(
 train_dir,
 target_size=(128, 128),
 batch_size=32,
 class_mode='categorical'
)
val_generator = val_datagen.flow_from_directory(
 val_dir, target_size=(128,
 128), batch_size=32,
 class_mode='categorical'
)
# Define CNN model model
= Sequential([
 Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
 MaxPooling2D(pool_size=(2, 2)),
 Conv2D(64, (3, 3), activation='relu'),
 MaxPooling2D(pool_size=(2, 2)),
 Conv2D(128, (3, 3), activation='relu'),
 MaxPooling2D(pool_size=(2, 2)),
 Flatten(),
 Dense(128, activation='relu'),
 Dropout(0.5),
 Dense(train_generator.num_classes, activation='softmax')
])
# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model history
= model.fit(
 train_generator,
 epochs=10,
```

```
 validation_data=val_generator
)
# Save the model
model.save('/content/drive/MyDrive/vehicle_model.h5')
# Plot training history
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label = 'Validation Accuracy')
plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.ylim([0, 1])
plt.legend(loc='lower right') plt.show()
Inference (Testing on New Images)
from tensorflow.keras.preprocessing import image
import numpy as np # Load the saved model
model = tf.keras.models.load_model('/content/drive/MyDrive/vehicle_model.h5')
# Load and preprocess the test image
img_path = '/content/drive/MyDrive/vehicle_dataset/test_image.jpg'
img = image.load_img(img_path, target_size=(128, 128)) img_array
= image.img_to_array(img) / 255.0 img_array =
np.expand_dims(img_array, axis=0)
# Predict the class predictions =
model.predict(img_array) predicted_class =
np.argmax(predictions[0]) print(f'Predicted
class: {predicted_class}')
```

6. Result:

Fig 1. sliding window approach is used having width and height 100*100

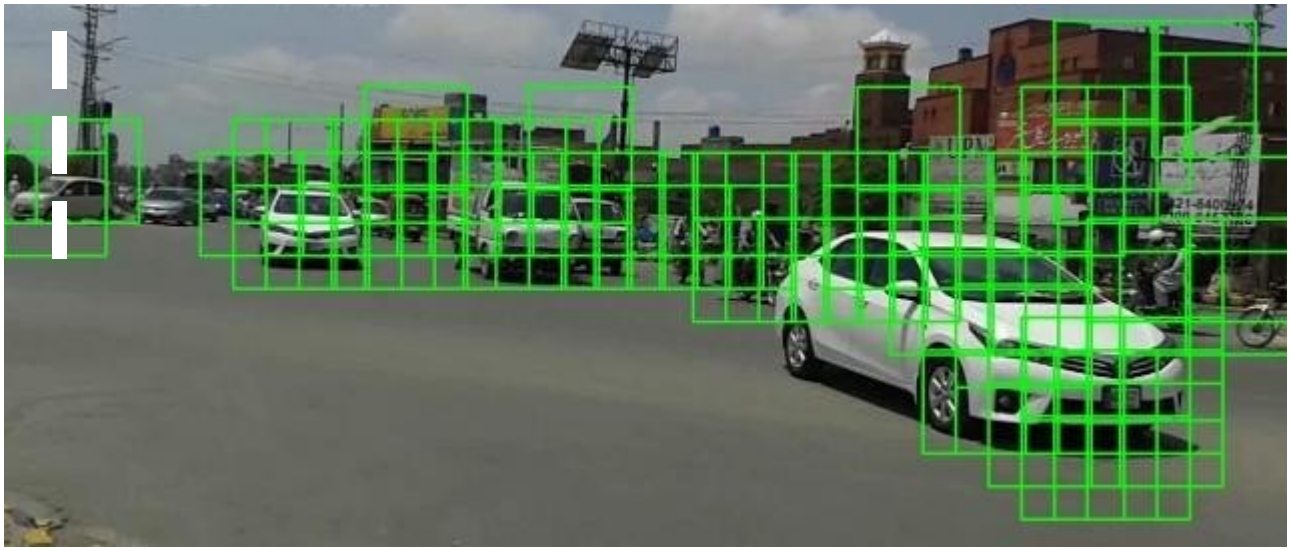


Fig 2. Than we perform the same function on our original Image and then perform testing

22BCS10806 TAPAN PADHI

Fig 3. we use group_rec function here and got following results

22BCS10806 TAPAN PADHI

Fig 4. detect vehicles on unseen data using cnn network

7. Learning Outcomes:

1. Choose the right AMI, instance type, and storage for deployment.
2. Manage AWS Free Tier resources efficiently to avoid unexpected costs.
3. Develop skills in monitoring instance performance and troubleshooting issues.

EXPERIMENT - 9

Student Name: Diwakar Kumar **UID:** 22BCS10849
Branch: BE-CSE **Section/Group:** IOT_640‘B’
Semester: 6th **Date of Performance:** 14/04/25
Subject Name: Foundation of Cloud IOT Edge ML **Subject Code:** 22CSP-367

1. **Aim: Automate quality inspection of products using cameras and edge computing.**

2. **Objective:** To design and implement an automated quality inspection system for products using cameras and edge computing.

3. **Pre requisites:**

Software Requirements:

1. Python (version 3.8 or above)
2. TensorFlow/Keras or PyTorch
3. OpenCV for image processing
4. Flask/Django for backend integration
5. MQTT or HTTP protocols for IoT data transfer

4. **Procedure:**

Step 1: Data Collection (Image Acquisition)

1. Cameras: Use high-quality cameras (e.g., industrial cameras or machine vision cameras) to capture product images. Depending on the application, cameras can be positioned at various points along the production line.
2. Lighting: Proper lighting is essential to ensure clear and consistent image capture. Lighting can be adjusted to reduce shadows and enhance defect visibility.
3. Trigger Mechanism: Use sensors (like proximity sensors or conveyors) to trigger the camera when a product passes through.

Step 2: Preprocessing the Images

4. Image Preprocessing: Raw images may need preprocessing to enhance features and remove noise.
5. Resizing to a fixed dimension (e.g., 224x224 pixels).
6. Normalization to scale pixel values.
7. Data Augmentation (if training a model) to simulate different conditions such as rotations or lighting variations.

Step 3: Defect Detection Model

8. Convolutional Neural Networks: A CNN is a deep learning model ideal for image classification tasks, like defect detection. You can either train your own CNN or use pre-trained models for defect classification.

9. Pre-trained Models for Feature Extraction: Use pre-trained models like ResNet or VGG16 for feature extraction and fine-tune them on your dataset if you have a labeled dataset of defective vs. non-defective products.
10. Inference on Edge Devices: The trained model is deployed to an edge computing device like a Raspberry Pi, NVIDIA Jetson, or an industrial-grade embedded system for inference.

Step 4: Edge Computing

11. Real-Time Processing: Edge computing allows processing images locally to minimize latency and avoid transferring large amounts of data to the cloud. This is particularly important for real- time applications where immediate decisions are needed (e.g., stopping the production line or rejecting defective products).
12. Hardware Selection: Use edge devices like Raspberry Pi, NVIDIA Jetson, or Intel NUC, depending on the complexity of your model and the required processing power.
13. Model Optimization: For efficient inference on edge devices, models can be optimized by:
    - Quantization: Reducing the precision of model weights (e.g., from float32 to int8) to speed up inference without sacrificing much accuracy.
    - Model Pruning: Removing unnecessary neurons to reduce the size of the model

Step 5: Integration with Actuators

14. Rejecting Defective Products: Once the defect is detected, the system can trigger an actuator (e.g., robotic arm, conveyor belt diverter) to reject or separate defective products from the production line.
15. Alerting Operators: When a defect is detected, the system can send alerts to factory operators or managers via SMS, email, or the IoT dashboard.

5. Implementation / Code:

```
import cv2 import numpy as np import tensorflow as tf from tensorflow.keras.applications import VGG16 from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Flatten

# Load and preprocess image def preprocess_image(image_path):
    image = cv2.imread(image_path)
    image_resized = cv2.resize(image, (224, 224))
    image_normalized = image_resized / 255.0 return
    np.expand_dims(image_normalized, axis=0)

# Load VGG16 pre-trained model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Define custom model for defect classification
model = Sequential([ base_model,
    Flatten(),
    Dense(512, activation='relu'),
```

```
Dense(2, activation='softmax') # 2 classes: defect or no defect
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Load the trained model model =

tf.keras.models.load_model("defect_detection_model.h5")

# Predict function def
predict_defect(image_path): image =
preprocess_image(image_path) prediction
= model.predict(image) class_index =
np.argmax(prediction) if class_index == 0:
print("Product is defective")
send_alert("Defective product detected!")
trigger_actuator()
    else:
        print("Product is not defective")

# Example usage predict_defect("product.jpg")
```

6. Screenshot:

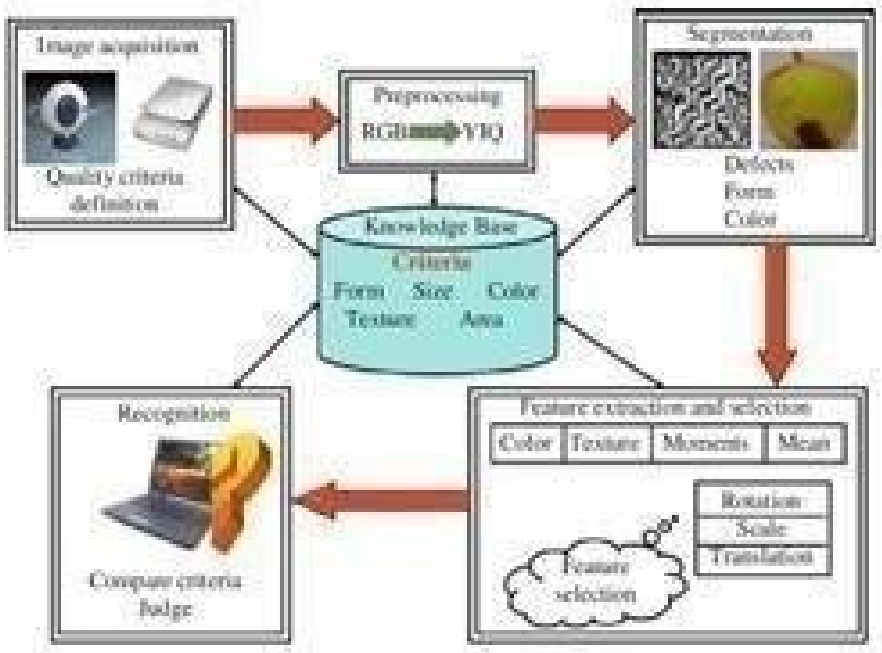


Fig 1.1 Fundamental steps of a computer vision for digital image processing

COMPUTER SCIENCE & ENGINEERING

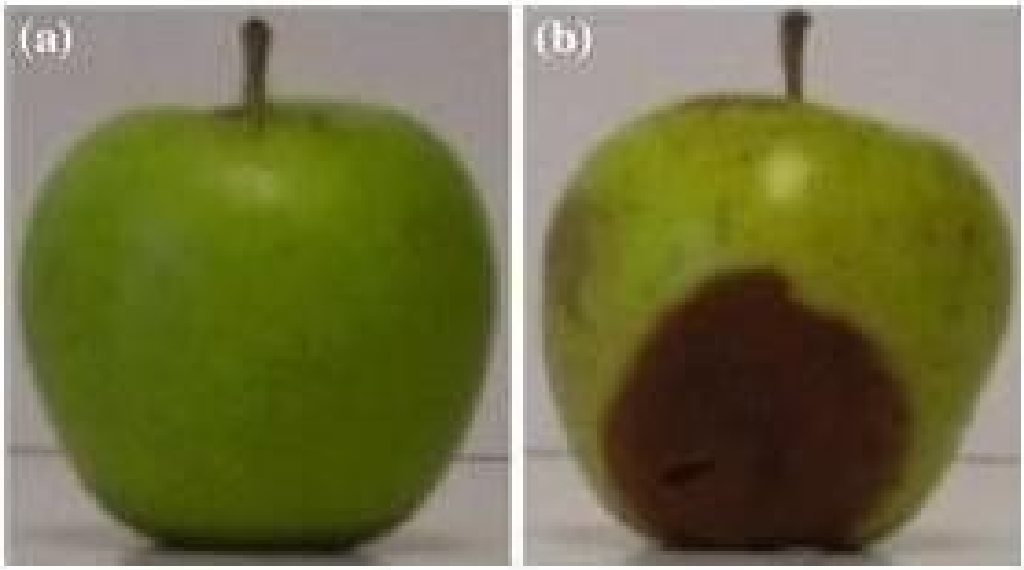


Fig 1.2 Examples of category extra golden apples. A) good quality apple, and B) Bad quality apples

7. Conclusion:

In this experiment, we explored essential techniques for image preprocessing, transfer learning, and model deployment to build an effective deep learning pipeline for product defect classification. By leveraging OpenCV and NumPy, we efficiently loaded, resized, and normalized images to ensure optimal input for deep learning models.

Using transfer learning with a pre-trained VGG16 model, we extracted meaningful features and customized the network to differentiate between defective and non-defective products, improving classification accuracy. Finally, we successfully deployed the trained TensorFlow model, demonstrating its capability to make real-time predictions and trigger automated responses, such as alerts or actuator controls. This workflow highlights the practical implementation of deep learning for industrial automation, enhancing defect detection efficiency and reliability.

Experiment 8

Student Name: Tapan Kumar Padhi **UID:** 22BCS10806
Branch: CSE **Section/Group:** 22BCS_IOT-602/A **Semester:** 6th **Date of Performance:** 18-03-25
Subject Name: Computer Graphics **Subject Code:** 22CSH-352

1. **Aim: a).** Apply the Cohen-Sutherland Line Clipping algorithm to clip a line intersecting at one point with a given window.
   b). Apply the Cohen-Sutherland Line Clipping algorithm to clip a line intersecting at two or more points with a given window

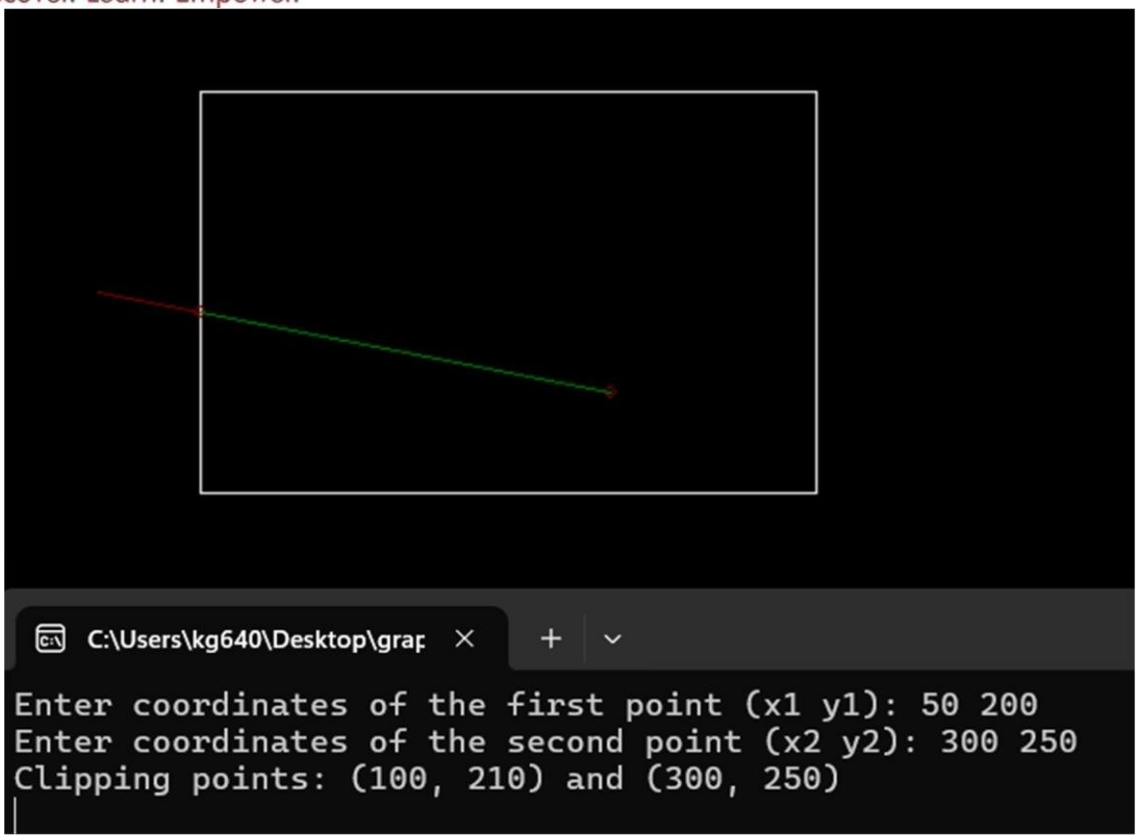
2. **Objective:** To clip a line intersecting at a single point and two or more points with a window using the Cohen-Sutherland Line Clipping algorithm.

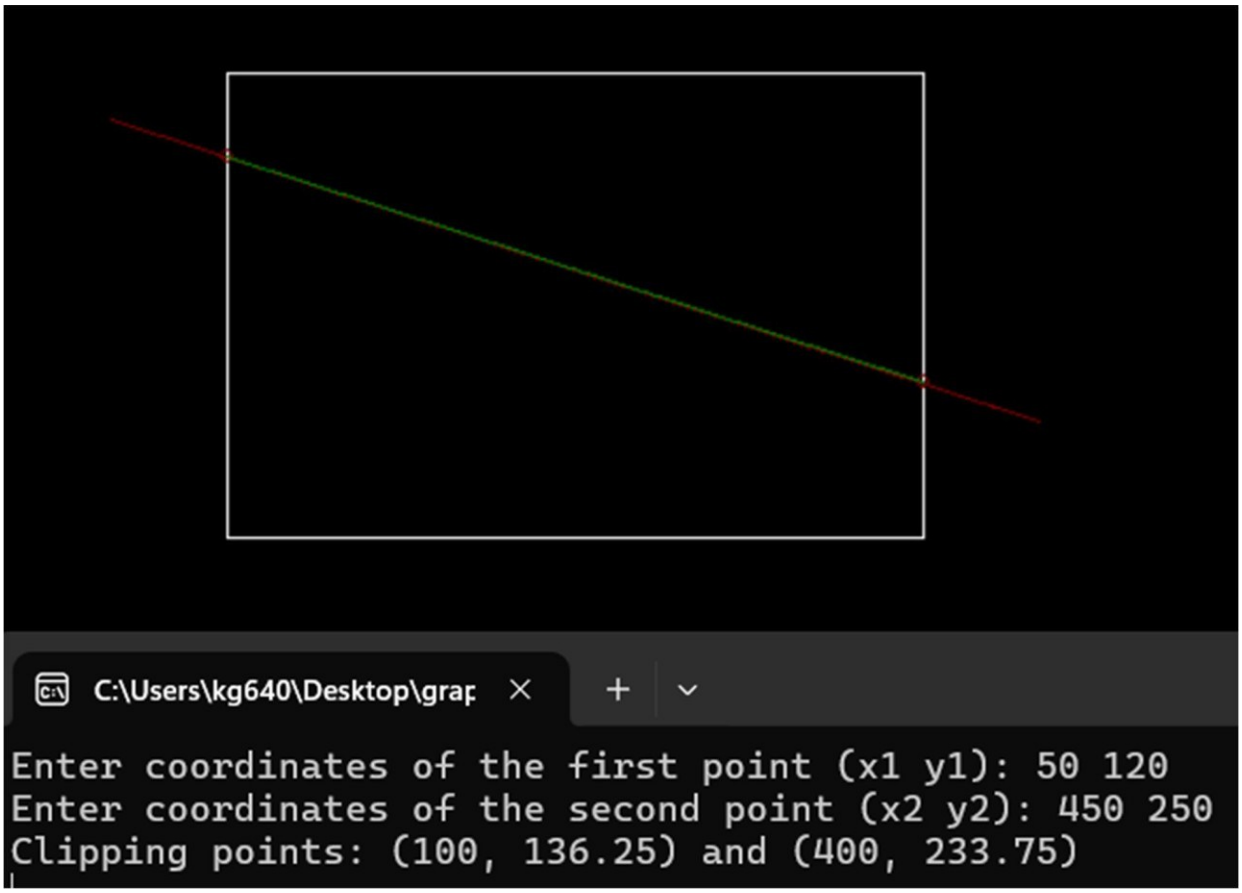
3. **Implementation/Code:** #include <iostream> #include <graphics.h> using namespace std;

```
const int x_min = 100, y_min = 100, x_max = 400, y_max = 300;
void computeCode(float x, float y, int code[4]) {
   code[0] = (x < x_min) ? 1 : 0; code[1] = (x >
   x_max) ? 1 : 0; code[2] = (y > y_max) ? 1 : 0;
   code[3] = (y < y_min) ? 1 : 0;}
bool isOutside(int code1[4], int code2[4]) {
   for (int i = 0; i < 4; i++) { if (code1[i] &
   code2[i]) return true;}
   return false;}
void cohenSutherlandClip(float x1, float y1, float x2, float y2) { int start[4], end[4];
   computeCode(x1, y1, start); computeCode(x2, y2, end); if (!isOutside(start, end))
   { if ((start[0] == 0) && (start[1] == 0) && (start[2] == 0) && (start[3] == 0) &&
   (end[0] == 0) && (end[1] == 0) && (end[2] == 0) && (end[3] == 0)) {
   setcolor(WHITE); line(x1, y1, x2, y2);
      } else { float m = (x2 - x1 == 0) ? 0 : (y2 - y1) / (x2
         - x1); float cx1 = x1, cy1 = y1, cx2 = x2, cy2 =
         y2; if ((start[2] == 0) && (start[3] == 1)) { cx1 =
         x1 + (y_min - y1) / m; cy1 = y_min;} if ((end[2]
         == 0) && (end[3] == 1)) {
            cx2 = x2 + (y_min - y2) / m;
            cy2 = y_min;}
         if ((start[2] == 1) && (start[3] == 0)) {
            cx1 = x1 + (y_max - y1) / m; cy1 =
            y_max;}
         if ((end[2] == 1) && (end[3] == 0)) { cx2
            = x2 + (y_max - y2) / m;
            cy2 = y_max;}
```

```
            if ((start[1] == 0) && (start[0] == 1)) {
               cy1 = y1 + m * (x_min - x1); cx1 =
               x_min;}
            if ((end[1] == 0) && (end[0] == 1)) {
               cy2 = y2 + m * (x_min - x2); cx2 =
               x_min;}
            if ((start[1] == 1) && (start[0] == 0)) {
               cy1 = y1 + m * (x_max - x1); cx1 =
               x_max;}
            if ((end[1] == 1) && (end[0] == 0)) {
               cy2 = y2 + m * (x_max - x2); cx2
               = x_max;}
            cout << "Clipping points: (" << cx1 << ", " << cy1 << ") and (" << cx2 << ", " << cy2 <<
")"    <<    endl;    setcolor(RED);
            circle(cx1,       cy1,       3);
            circle(cx2,       cy2,       3);
            setcolor(GREEN);
            line(cx1,       cy1,       cx2,
            cy2);}}}
int   main()   {   int   gd   =
   DETECT,              gm;
   initgraph(&gd,      &gm,
   ""); setcolor(WHITE);
   rectangle(x_min, y_min, x_max, y_max);
   float x1, y1, x2, y2; cout << "Enter coordinates of the
   first point (x1 y1): "; cin >> x1 >> y1;
   cout << "Enter coordinates of the second point (x2 y2): ";
   cin >> x2 >> y2; setcolor(RED); line(x1, y1, x2, y2);
   delay(2000);
   cohenSutherlandClip(x1, y1, x2, y2);
   delay(5000); getch(); closegraph();
   return 0;}
```

4. Output


C:\Users\kg640\Desktop\grap
Enter coordinates of the first point (x1 y1): 50 200
Enter coordinates of the second point (x2 y2): 300 250
Clipping points: (100, 210) and (300, 250)


C:\Users\kg640\Desktop\grap
Enter coordinates of the first point (x1 y1): 50 120
Enter coordinates of the second point (x2 y2): 450 250
Clipping points: (100, 136.25) and (400, 233.75)

5. Learning Outcome

- Understanding Line Clipping – Learned Cohen-Sutherland algorithm, region codes, and clipping logic.
- Debugging C Graphics – Fixed division by zero, logical errors, and improved code efficiency.
- Graphics Implementation – Used Turbo C++ functions to create and modify viewport-based line rendering.

Experiment 9

Student Name: Tapan Kumar Padhi UID: 22BCS10806 Branch: B.E CSE
Section/Group: IOT_602/ A Semester: 6th Date of Performance:03-04-25 Subject Name: Computer Graphics Subject Code: 22CSH-352

1. **Aim:** Demonstrate the result of window-to-viewport transformation by implementing and visualizing the process.

2. **Objective:** To Demonstrate the result of window-to-viewport transformation by implementing and visualizing the process.

3. **Implementation/Code:**

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>

void main() {
   int gd=DETECT, gm;
   initgraph(&gd, &gm, ""); // Use empty string instead of NULL clrscr();
   // Place after initgraph()

   float wxmin=10, wxmax=150, wymin=10, wymax=250;
   float vxmin=200, vxmax=600, vymin=10, vymax=250;
   int wx1=30, wy1=50, wx2=100, wy2=180;

   // Draw window and viewport
   rectangle(wxmin, wymin, wxmax,
   wymax); rectangle(vxmin, vymin, vxmax,
   vymax);

   // Scaling factors
   float sx = (vxmax - vxmin) / (wxmax - wxmin); float
   sy = (vymax - vymin) / (wymax - wymin);

   // Draw original line line(wx1,
   wy1, wx2, wy2); // Transform
```

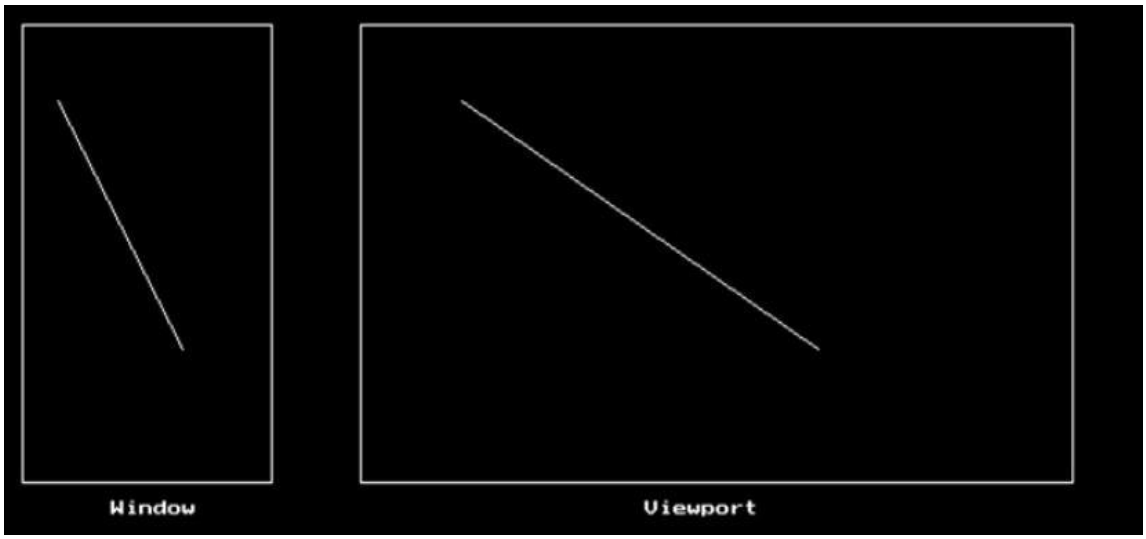
```
coordinates float vx1 = sx *
(wx1 - wxmin) + vxmin; float
vy1 = sy * (wy1 - wymin) +
vymin; float vx2 = sx * (wx2 -
wxmin) + vxmin; float vy2 =
sy * (wy2 - wymin) + vymin;

// Draw transformed line in viewport line(vx1,
vy1, vx2, vy2);

// Labels
outtextxy(60, 260, "Window"); outtextxy(360,
260, "Viewport");

getch();
closegraph(); // Properly exit graphics mode }
```

4. Output



5. Learning Outcome

- Learned how to transform coordinates from a window to a viewport.
- Practiced **rectangle(), line(),** and **outtextxy()** for visualization.
- Implemented formulas to scale and position graphical objects.